Počítačové šachy

Otakar Trunda

Hraní her obecně

Hra je definovaná pomocí:

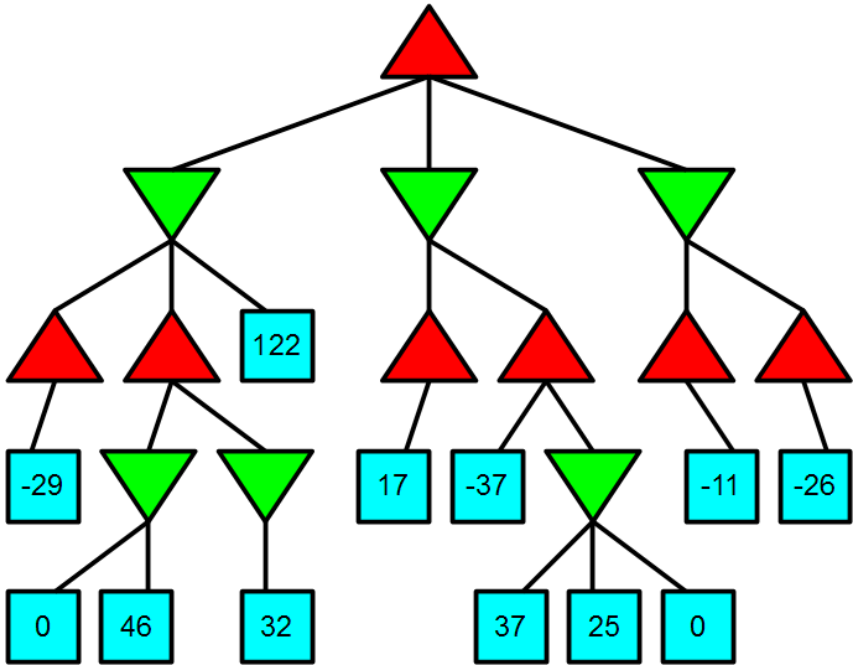
- Počáteční situace
- Funkce vracející množinu přípustných tahů v každé situaci
- Ohodnocení koncových stavů

Našim cílem je najít strategii pro úspěšné hraní hry.

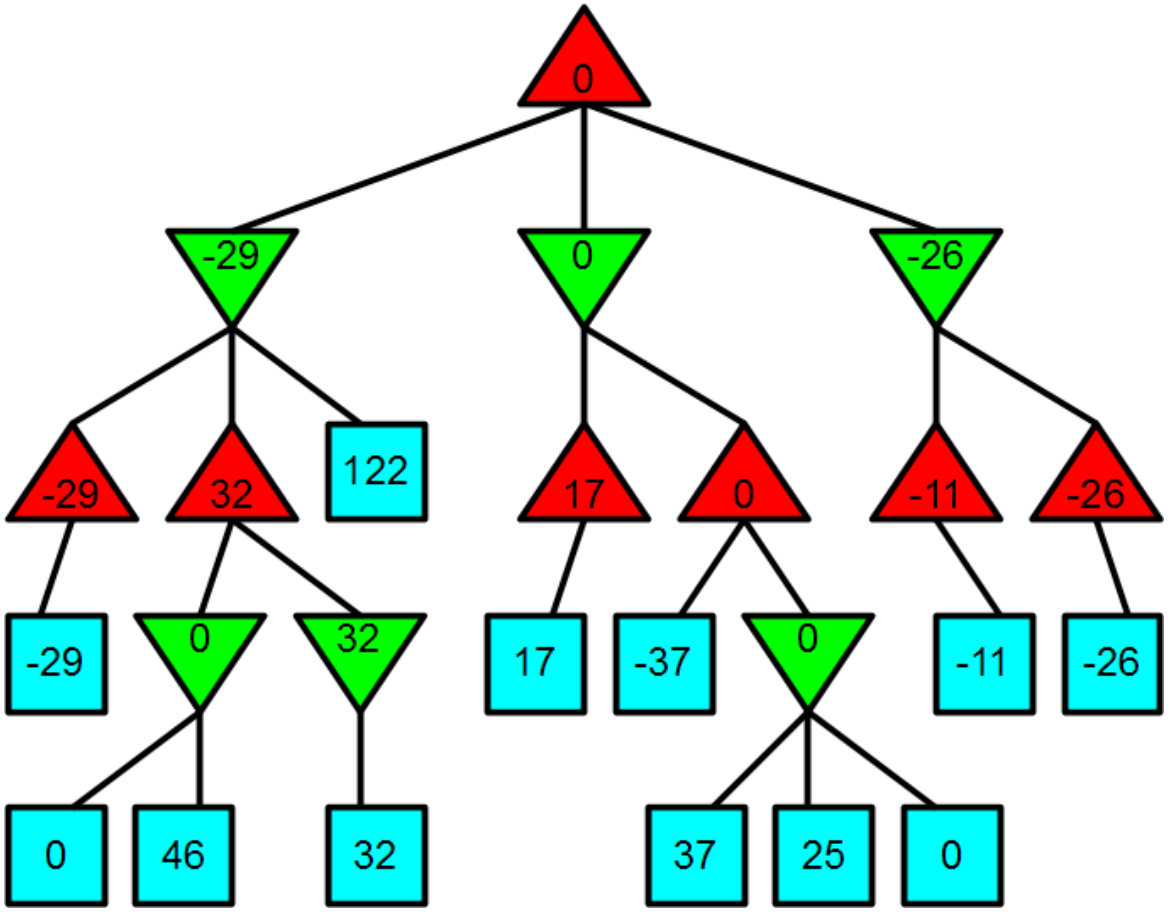
Algoritmus Minimax

- Nejjednodušší algoritmus
- Najde vždy optimální strategii
- Prochází celý strom hry
- Časová složitost

  $O\left(b^n\right)$ kde b ≈ 35-38



Minimax - výsledek



α-β prořezávání

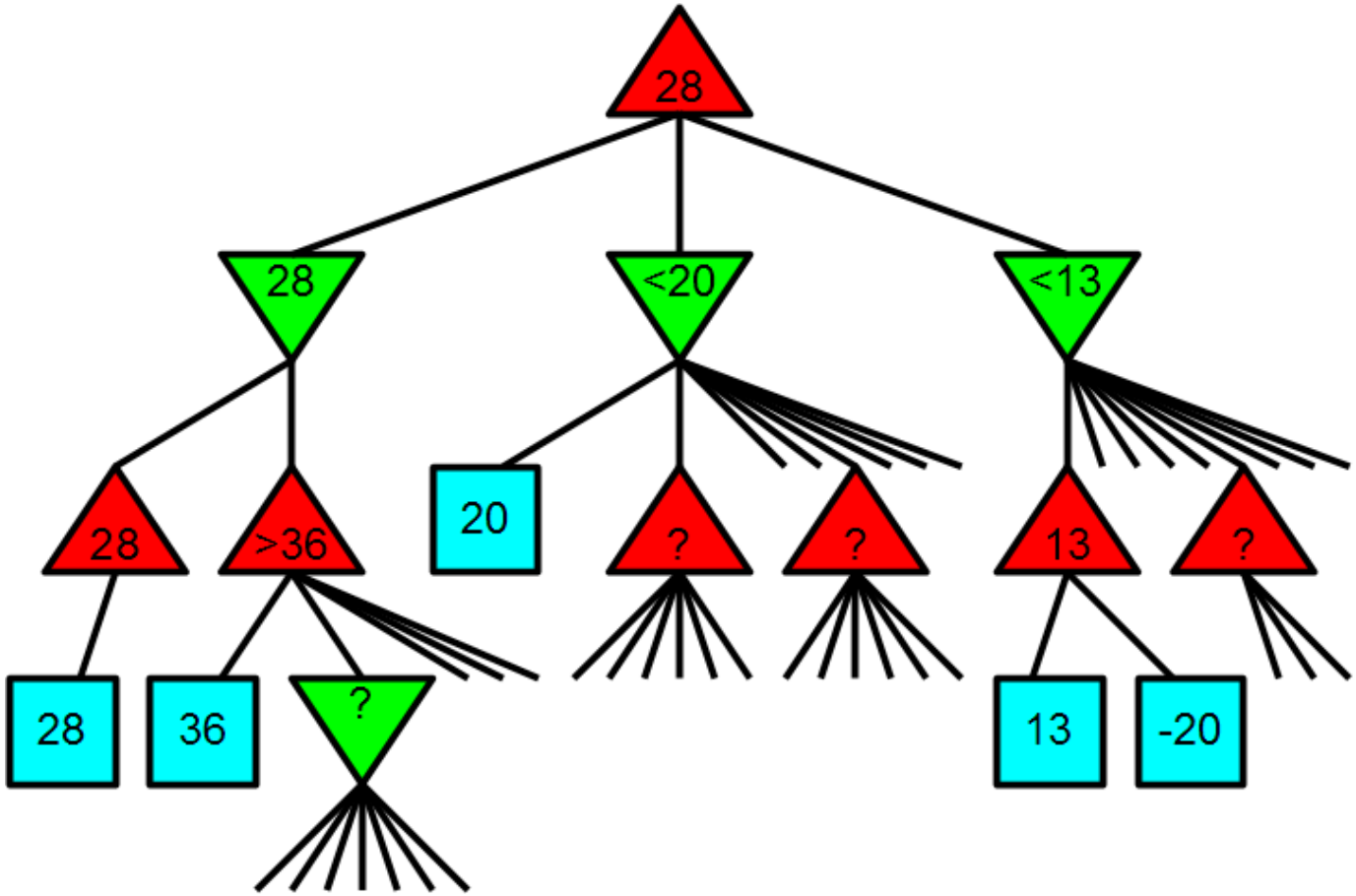
- Pozorování:

Minimax hodnota kořene **nezávisí** na hodnotách **všech** vrcholů stromu

- Algoritmus α-β prořezávání:

Stejný princip jako minimax, ale prohledává jen to, co je třeba

α-β prořezávání - příklad



α-β prořezávání - kód

```
function SearchMax(node, α, β)
    if node is a terminal node
        return the value of node
    best = -∞
    foreach child of node
        score := SearchMin (child, α, β)
        best := max(best, score)
        if best ≥ β
                return best
        α := max(α, best)
    return best
```

```
function SearchMin(node, α, β)
    if node is a terminal node
        return the value of node
    best = ∞
    foreach child of node
        score := SearchMax (child, α, β)
        best := min(best, score)
        if best ≤ α
                return best
        β := min(β, best)
    return best
```

α-β kvíz

- α-β najde stejný nejlepší tah jako Minimax?

  ANO

- α-β ohodnotí nejlepší tah stejně jako Minimax?

  ANO

- α-β ohodnotí všechny tahy stejně jako Minimax?

  NE

- α-β zachová pořadí tahů stejně jako Minimax?

  NE

Efektivita α-β prohledávání

- Nejlepší případ:

  Efektivní větvící faktor $\sqrt{b}$

  Časová složitost $O\left(b^{n/2}\right)$

  Dvojnásobná hloubka oproti Minimaxu

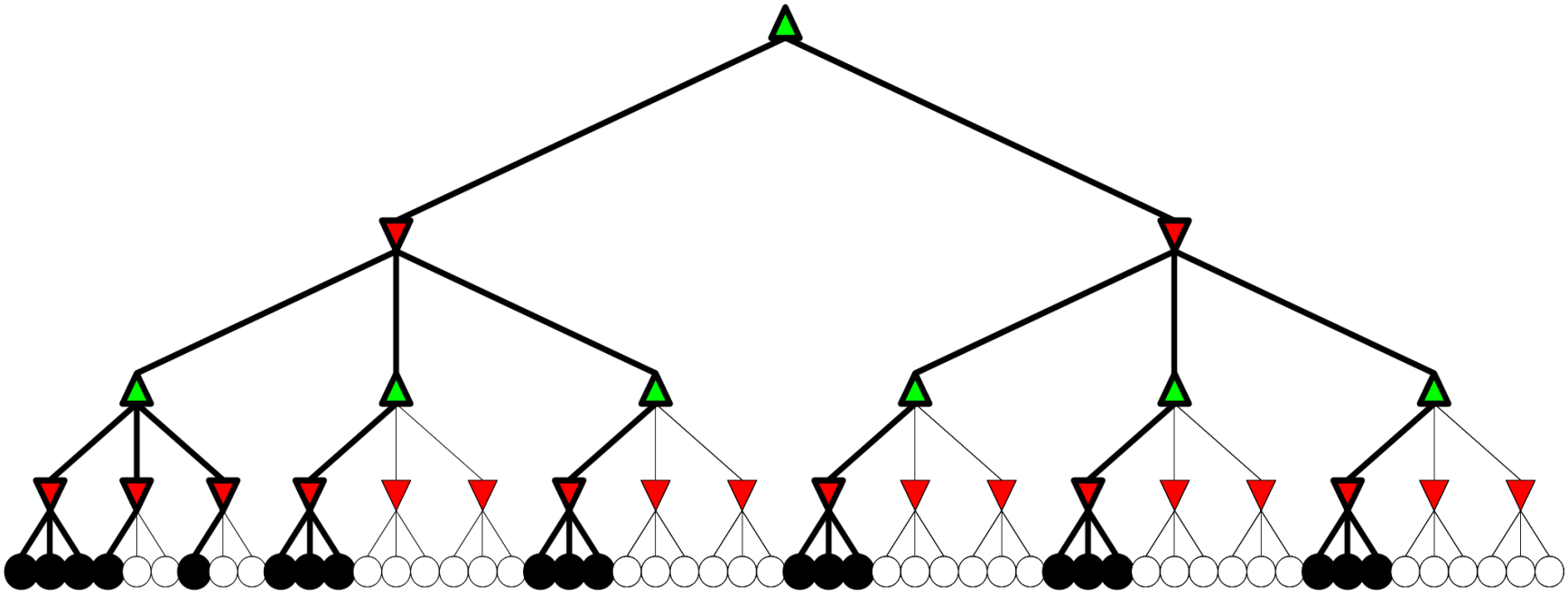
- Nejhorší případ:

  Žádné prořezávání, větvící faktor b

  Časová složitost $O\left(b^{n}\right)$ + režie navíc

  Pomalejší než Minimax

α-β – nejlepší případ



Efektivita prořezávání

- Množství prořezaných větví závisí na velikosti intervalu <α, β>
- Zvyšování efektivity:
  - Metoda okénka
  - Úderné tahy a Killery
  - Nulové okno - algoritmy typu Scout
  - Null-move heuristika
  - Futility pruning, Delta pruning a další

Killer heuristika

- Killer: dobrý tah, který (hodně) zmenšil interval <α, β>
- Během prohledávání udržujeme tabulku killerů, tyto tahy zkoušíme jako první
- Zobecnění: History heuristika
  - Kdykoliv najdeme killer, inkrementujeme hodnotu čítače u tohoto tahu
  - Tuto hodnotu použijeme pro řazení tahů při dalším prohledávání

Metoda okénka

- Každý killer zmenší interval <α, β>
- Na začátku nastaven na <-∞, ∞>
  prohledáváme mnoho špatných tahů
- Metoda okénka:
  - Předem odhadneme hodnotu nejlepšího tahu
  - Interval nastavíme kolem této hodnoty
  - Při špatném nastavení, musíme prohledávat znovu!

Metoda okénka

Při prohledávání s nastaveným intervalem $<\alpha, \beta>$ mohou nastat 3 případy:

- Nalezneme řešení v intervalu

  OK

- Nalezneme pouze řešení $< \alpha$ tzv. Fail-low

  Prohledáme znovu s intervalem $<-\infty, \alpha>$

- Nalezneme pouze řešení $> \beta$ tzv. Fail-high

  Prohledáme znovu s intervalem $< \beta, \infty >$

Nulové okno

- Prohledává s oknem velikosti <α, α+1>
- Pouze testuje, zda je skutečná hodnota menší nebo větší než α
- Algoritmus Scout:
  - Heuristicky odhadne nejlepší tah
  - Spočítá jeho hodnotu
  - Pomocí nulového okna dokáže, že ostatní tahy jsou horší

Algoritmus NegaScout

```
function negascout(node, depth, α, β)
    if node is a terminal node or depth = 0
        return the heuristic value of node
    b := β                              // initial window is (-β, -α)
    foreach child of node
        score := -negascout (child, depth - 1, -b, -α)
        if α < score < β and child is not first child
                                        // check if null-window failed high
            score := -negascout(child, depth - 1, -β, -α)        // full re-search
        α := max(α, score)
        if α ≥ β
            return α                    // Beta cut-off
        b := α + 1                      // set new null window
    return α
```

Null-move heuristika

- Pozorování: Téměř vždy je lepší zahrát tah, než nedělat nic
- Null-move heuristika - myšlenka:
  - Pokud po vynechání tahu je hodnota pozice větší než β, pak bude větev stejně odřezaná
  - Hodnotu podstromu stačí spočítat do nějaké malé hloubky (typicky 3)
- Pokud heuristika selže, vrátí alespoň killer pro další hladinu
- Nelze použít vždy

Futility Pruning

- Jedná se o tzv. Forward pruning
- Ořezává větve, které mají malou naději překonat hodnotu α
- Odhaduje maximální možnou změnu hodnoty pozice za 1 tah
- Příklad: prořezává větve 1 tah před limitní hloubkou pokud

  hodnotaPozice < α - hodnotaNejsilnejsiFigury - konstanta
- Hrozí ztráta optimálního řešení, nutno používat opatrně

Nedokonalé strategie

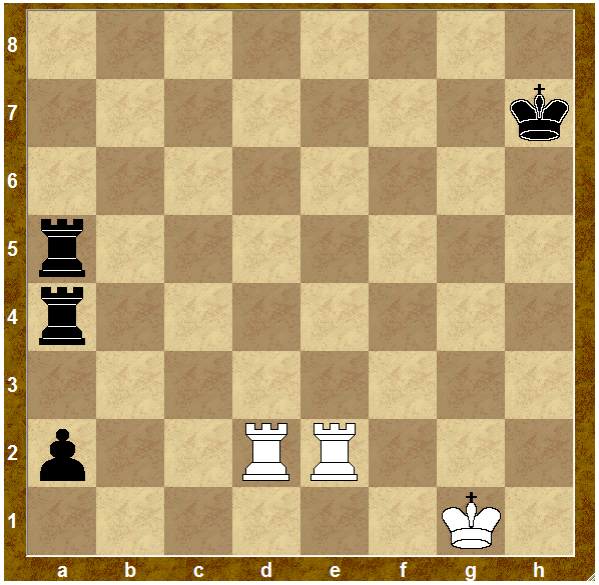
- Z časových důvodů nelze prohledat celý strom
- Prohledáváme jen do určité hloubky, kde použijeme heuristický odhad výsledku
- Výpočet skončí v požadovaném čase, ale získaná strategie nemusí být optimální

Další techniky pro zvýšení výkonu

- Singulární prodloužení a Quiescence search jako řešení problému horizontu
- Databáze zahájení a koncovek
- Transpoziční a zamítací tabulky
- Kombinace popsaných metod pomocí Iterativního prohlubování

Problém horizontu (1)

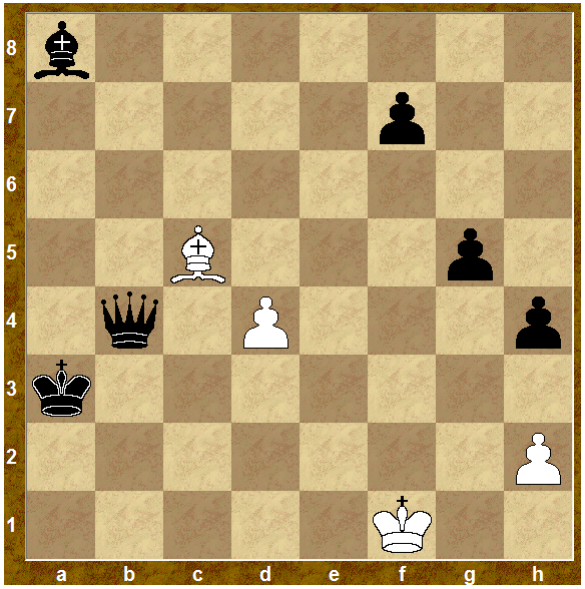
- Prohledání tahu za limitní hloubkou může dramaticky změnit hodnotu pozice
- Příklad:



Hloubka	Tahy	Hodnota
1	1.Vxa2	0
2	1.Vxa2 Vxa2	-500
3	1.Vxa2 Vxa2 2.Vxa2	0
4	1.Vxa2 Vxa2 2.Vxa2 Vxa2	-500

Problém horizontu (2)

- Odsouvání nevyhnutelné události za horizont (často pomocí obětí materiálu)
- Příklad:



Snaha oddálit ztrátu dámy:

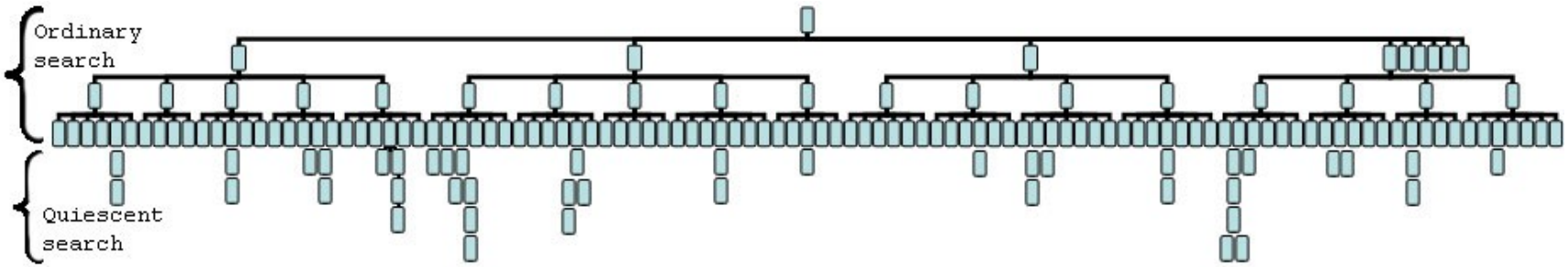
1. … Sg2+ 2.Kxg2 h4-h3+

3. Kxh3 g5-g4+ 4.Kxg4 atd.

Problém horizontu - řešení

Quiescence search:

- Prohledáváme i za limitní hloubku, ale pouze vybrané tahy
- Prohledávání končí, až když se pozice uklidní



Quiescence search

- Pozice je typicky považována za neklidnou pokud
  - je možné sebrat nějaký kámen
  - je možné proměnit pěšce
  - je možné dát šach
- Speciální případ metody zvané Singulární prodloužení
- Mnoho dalších modifikací např. Botvinnik-Markoff Extension

Databáze zahájení a koncovek

- Mohou výrazně zlepšit výkonnost programu
- Částečně eliminují problém s různými prioritami v zahájení a koncovce
- Zahájení:
  - Praxí ověřené varianty, několik desítek tahů
- Koncovky:
  - Optimální strategie až do matu
  - Nejdelší vyřešená koncovka: 7 kamenů, lze vynutit mat za 517 tahů

Transpoziční tabulky

- Na stejný uzel můžeme během prohledávání narazit vícekrát
- Jednou spočítané hodnoty lze uložit a později znovu použít
- Velmi efektivní zejména při použití iterativního prohlubování
- U prozkoumaného uzlu můžeme uložit:
  - Výsledek statické ohodnocovací funkce
  - Minimax hodnotu podstromu do určité hloubky
  - Seznam přípustných tahů v dané pozici (seřazený od nejlepších) a další informace

Iterativní prohlubování

- Namísto prohledání do hloubky *n*, hledáme postupně do hloubky 1,2,3,…,n
- Asymptoticky stejná časová složitost
  - V praxi mírně vyšší, ale lze výrazně zlepšit pomocí transpozičních tabulek
- Výpočty z předchozích vrstev lze výhodně použít pro heuristické odhady
  - Metoda okénka – odhad hodnoty pozice
  - Killery
  - Pořadí tahů pro Scout, atd…

Statická ohodnocovací funkce

- Výrazně ovlivňuje herní sílu enginu
- Měla by odpovídat skutečné ceně pozice
- Typicky se kombinují materiální i poziční faktory
- Mnoho ručně nastavených parametrů

Figura	Hodnota
Dáma	900-950
Věž	500
Střelec	300-310
Jezdec	300
Pěšec	100

Vlastnost pozice	Bonus
Dvojpěšec	-10
2 věže	+10-20
2 střelci	+20-30
Dáma+Jezdec	+10-20
Postouplý Pěšec	+30-100

Netradiční přístupy

- Nové možnosti při programování šachových enginů
- Využití evolučních algoritmů a neuronových sítí
- Nevhodné jako úplná náhrada prohledávacího algoritmu
- Lze použít pro nalezení kvalitní ohodnocovací funkce
- Velké časové nároky, proto se často používá zjednodušená doména (např. koncovka)

Projekt NeuroChess

- Snaží se naučit neuronovou síť na ohodnocování pozice
- Učí se podle konečného výsledku partie
- Sofistikované předzpracování vstupních dat, propracovaný algoritmus učení
- Přesto učení trvalo 2 týdny na 20 počítačích, poměrně slabé výsledky

Řešení koncovky K+V x K pomocí neuronové sítě

- Bez prohledávacího algoritmu
- Snaha klasifikovat pozici podle počtu tahů zbývajících do matu
- Učení podle databáze koncovek
- Dosažený výsledek:
  - 3 skryté vrstvy s celkem 82 neurony
  - doba učení 4 hodiny
  - úspěšnost klasifikace přes 80%

Děkuji za pozornost